IDNoise: Resource-Aware Machine Learning-Based Noise and SNR Detection in Electrocardiogram Signals

Soheil Khooyooz*[1], Vimala Bauer*[1], Anice Jahanjoo[2], Mostafa Haghi[1], Amin Aminifar[1], Nima TaheriNejad[1,2]

***Abstract*— Noise is a significant challenge in wearable Electrocardiography (ECG), as it can distort the ECG waveform and lead to inaccurate signal interpretation. Among noise sources, Motion Artifcats (MA)s are particularly difficult to mitigate due to their unpredictable time-frequency characteristics. Unlike electromagnetic interference, MAs often overlap with critical spectral components of the ECG signal, making their reduction in preprocessing especially challenging. In this work, we propose IDNoise, a Machine Learning (ML)-based approach for detection and identification of noise in ECG recordings. Our method leverages a comprehensive feature set, including morphological, statistical, and concatenated features, to train ML models capable of distinguishing various noise types and estimating their Signal to Noise Ratio (SNR). We evaluate the proposed algorithms in terms of execution time, energy consumption, and memory usage, which are critical resource usage metrics when designing solutions for wearable devices. We demonstrate that IDNoise, when using concatenated features, can detect the noise type in binary classification with an accuracy of 80.52% and an F1-score of 80.44%. It can also detect the noise type in 4-class classification with an accuracy of 67.91% and an F1-score of 67.89%, and identify the SNR level in 7-class classification with an accuracy of 44.80% and an F1-score of 44.57%. While the use of concatenated features necessitates, during the feature extraction phase, an increase in computational overhead of a factor up to 7.5 times (execution time, energy consumption, and memory usage). The computational overhead for prediction and loading the models remain comparable to that of individual features.**

I. Introduction

Wearable healthcare technology is bringing about a notable transformation in healthcare paradigms. Such systems can be used for continuous monitoring of individuals, which provides a more comprehensive and real-time view of their health [1], [2]. By adopting wearable technology, healthcare professionals can remotely monitor patients, particularly those with chronic conditions, without the need for frequent in-person visits. Exploiting the data collected by wearables supports personalized services and tailored healthcare interventions that can lead to more effective and efficient healthcare outcomes [3], [4].

Wearable systems enhance early detection of health issues by continuously monitoring subtle changes in metrics, enabling timely medical interventions and preventive measures, which can reduce healthcare costs through remote monitoring and early interventions [5]–[9]. These devices, particularly through real-time tracking of vital signs like Electrocardiogram (ECG) signals, play a crucial role in detecting and managing cardiovascular conditions, offering early warnings for irregular heart rhythms and other health concerns [10].

However, various types of noise at different intensities corrupt the ECG signal, making the output of wearable devices unreliable. Moreover, efficient execution time is crucial for timely interventions [11], [12]. Memory management is equally important, as inefficient memory usage can compromise data integrity, reduce storage capacity, and increase energy consumption [13], [14]. Additionally, power shortages can disrupt operations, potentially causing missed critical health data, which can have significant consequences [15]. The strain on hardware due to poor energy management and prolonged device use can further reduce the durability of wearable devices, leading to premature failure [11], [16].

In [17], the noise in ECG recordings is classified into four categories based on their origins. These four classes are outlined as: i) Electrode Motion (EM): This noise emerges from intermittent mechanical forces acting on the electrode. ii) Baseline Wander (BW): This noise originates from the motion of the subject or the leads. iii) Muscle Artifact: This noise arises from muscle activity, such as contractions or movement. iv) Power Line Interference (PLI): This noise results from the electromagnetic interference of the alternating supply [18]. As discussed by the authors in [17], PLI can be easily removed using a digital filter and can be ignored.

In this work, we refer to muscle artifact and EM as Motion Artifact (MA). Compared to minimizing electromagnetic interference, reducing MAs is more challenging due to their unpredictable time-frequency characteristics. Adaptive filtering is widely used [19]–[21], although effective; but it is difficult to determine if the noise estimation from the reference signal converges accurately to the actual noise in the recorded signal. Incorrect convergence can remove random components and distort the ECG waveform. For example, the authors in [22] proposed an adaptive MA reduction method using an improved Recursive Least Squares (RLS) adaptive filter, where the Impedance Pneumography (IP) signal serves as a reference for MAs in ECG signals. Their ECG-IP acquisition system, designed to measure both signals simultaneously, achieved high correlation ($\|r\| > 0.6$) and effectively reduced MAs with minimal signal distortion, as

* Both authors contributed equally to this research.

This work was partially supported by Hector Stiftung.

[1] S. Khooyooz, V. Bauer, M. Haghi, A. Aminifar, and N. TaheriNejad are with the Institute for Computer Engineering, Heidelberg University, Heidelberg, BW 69120, Germany.
{soheil.khooyooz, vimala.bauer, mostafa.haghi, amin.aminifar, nima.taherinejad}@ziti.uni-heidelberg.de

[2] A. Jahanjoo and N. TaheriNejad are with TU Wien, Institute of Computer Technology, Gusshausstrasse 27-29, Vienna, Austria.
{anice.jahanjoo, nima.taherinejad}@tuwien.ac.at
(corresponding author: Soheil Khooyooz)

demonstrated by real noisy ECG data analysis.

Nagai et al. in [23] proposed a Stationary Wavelet Transform (SWT) algorithm to remove MAs from ECG signals captured with non-contact electrodes. Applying the algorithm on the ECG signals with the MAs showed an improvement in correlation coefficients from 0.71 to 0.88 after artifact removal, demonstrating the algorithm's effectiveness. However, wavelet transform is limited by the need for optimal thresholds, which can vary and be difficult to obtain, leading to inconsistent performance.

The Empirical Mode Decomposition (EMD) and its variance also have been applied to noisy ECG signals to reduce the effect, however, MAs are correlated with both low- and medium-frequency Intrinsic Mode Functions (IMFs) in EMD. Therefore, EMD-based methods, such as the one in [24], [25], that do not effectively address medium-frequency IMFs may result in insufficient denoising.

Zou et al. in [26] proposed QRS detection based Motion Artifact Removal algorithm (QRSMR) that targets the QRS complexes in motion-contaminated ECG signals, preserving these key segments while filtering out noise. The approach detects and repairs irregular QRS complexes, removes baseline wander by interpolating between Q- and S-peaks, and applies a moving average to denoise intervals between complexes, retaining P- and T-waves. Testing on simulated noisy ECG signals showed an improvement in correlation with clean signals from 40% to nearly 80%, demonstrating the method's enhanced noise removal capabilities.

The authors in [27] proposed an automatic ECG noise detection and classification method using moving average filtering, feature extraction, and a multistage decision-tree algorithm. Dynamic amplitude range and autocorrelation peak features were used to detect and classify low-frequency (baseline wander, abrupt change) and high-frequency (power line interference, muscle artifacts) noise. Testing demonstrates an average sensitivity of 97.88%, positive productivity of 91.18%, and accuracy of 89.06%.

Addressing the challenges of wearable healthcare devices requires effective management of time, energy, and memory. Reducing energy consumption extends battery life and reduces hardware strain, enhancing device durability [11], [16]. Efficient memory management ensures data integrity and optimizes storage while minimizing energy consumption [13], [14]. Furthermore, memory-efficient algorithms improve data sharing and interoperability with healthcare systems [13]. Profiling execution time, energy consumption, and memory usage is key to improving monitoring performance, enhancing user experience, and ensuring wearable devices' long-term reliability and functionality. None of the available research has simultaneously addressed the detection of both noise types and SNR levels in ECG signals, along with performance profiling. In this paper, we use statistical and morphological features to detect noise types and SNR levels, while also profiling the performance of various adapted Machine Learning (ML) models and features. We comprehensively profile time, energy, and memory, providing a more holistic approach to improving wearable device performance.

The rest of this paper is organized as follows: Section II elaborates on the dataset and methodology. Section III reports and discusses the results. Finally, we conclude the paper in Section IV.

II. Materials and Methods

In this section, we elaborate on the dataset, methods, and algorithms we use for noise detection in ECG signals, along with the techniques used to measure the time, energy, and memory required to implement these algorithms.

A. Dataset

We use MIT-BIH Arrhythmia Dataset [28], [29] in our experiments to evaluate IDNoise. This dataset comprises 48 half-hour two-channel recordings of ECG signals from 47 subjects. In accordance with [30], we treat the recordings of this dataset as clean signals. We employ the WFDB software package [31] to contaminate the recording according to [17], [30]. Employing the default settings of the WFDB software and specifying the type and intensity of noise, starting after the initial 5 minutes of each recording, two-minute intervals of noise are introduced, alternating with two-minute clean segments. We then segment the signals into 10-second intervals (windows). Therefore, we have 4 classes of signals (including muscle artifact, EM, BW corrupted signals, and the clean signal), 7 classes of signals based on the SNR level (including -6 dB, 0 dB, 6 dB, 12 dB, 18 dB, 24 dB, and the clean signal), and 19 classes of signals based on both the type of noise and the SNR level (3 different types of noise, each with 6 different SNR levels, and the clean signal).

B. Feature Extraction

We extract different types of features from the data including statistical and morphological features. For the morphological, we use two approaches: 1) raw morphological features, 2) processed morphological feature.

1) Statistical Features

Statistical features provide insights into their distribution and variability. These features include the mean, variance, kurtosis, skewness, energy, entropy, and maximum autocorrelation. Additionally, we analyze the mean, variance, and maximum of the signal's histogram [32], [33].

2) Morphological Features

These features describe the shape, pattern, and specific attributes of signals. Since noise can corrupt the shape and pattern of the ECG signal, we use the vital-sqi [34] open-access Python toolbox to pinpoint the locations of peaks in the signal. These peak locations offer insights into the time intervals between peaks, which we regard as morphological features [32], [35].

1. Raw Morphological Features: The pattern of distances between detected peaks differs among various classes. For example, in signals affected by MA, some peaks might not be detected, leading to a unique distance pattern compared to other classes. To standardize the feature vectors, we perform the following steps: 1) Sort the features in each feature vector in descending order. 2) Compute the mean (μ) and standard deviation (σ) of the vectors. 3) Define the length of the

TABLE I: LIST OF FEATURES

Statistical Features	
Mean	Variance
Kurtosis	Skewness
Energy	Entropy
Maximum Auto-correlation	Histogram's Mean
Histogram's Variance	Histogram's Maximum
Raw RR Features	
Sorted RR intervals	
Processed RR Features (HRV Features)	
SDNN	RMSSD
PNN50	LF (Low Frequency)
HF (High Frequency)	LF/HF Ratio
SD1	SD2
Num_peaks	Mean_NNs

feature vectors as $l = \lfloor \mu \rfloor + \lfloor \sigma \rfloor$. 4) Zero-pad feature vectors shorter than l and truncate the last samples in feature vectors longer than l [32], [35].

2. Processed Morphological Features: Using the raw morphological features (RR-intervals, also known as NN intervals), we also calculate some Heart Rate Variability (HRV) features using pyhrv library in python [36] as follows: 1) SDNN (Standard Deviation of successive normal heartbeats or NN intervals): Measures overall heartbeat variability over a period. 2) RMSSD (root mean square of successive differences): Reflects short-term variations in heart rate, influenced by parasympathetic activity. 3) pNN50 (percentage of successive NN intervals differing by > 50 ms): Indicates the proportion of intervals with significant differences, related to parasympathetic activity. 4) LF (Low Frequency): Represents power in the low-frequency range (0.04 to 0.15 Hz) of the HRV spectrum, associated with both sympathetic and parasympathetic activity. 5) HF (High Frequency): Represents power in the high-frequency range (0.15 to 0.40 Hz) of the HRV spectrum, primarily linked to parasympathetic activity. 6) LF/HF Ratio: Indicates the balance between sympathetic and parasympathetic nervous system activity. 7) SD1 (Standard Deviation 1): Reflects short-term, calculated from the Poincaré plot. 8) SD2 (Standard Deviation 2): Reflects long-term HRV, also derived from the Poincaré plot. 9) Num_peaks (Number of Peaks): The count of detected peaks in the heart rate signal, indicating significant heart rate changes. 10) Mean_NNs (Mean of NN intervals): The average duration between successive R-wave peaks in the ECG, representing the average heart rate. After feature extraction, we normalize them to ensure a consistent scale across all features, which improves the performance and convergence of ML models. A summary of the statistical and morphological features used is provided in Table I.

C. Classification

In this study, we conduct the multi-class classifications as follows: i) classification of noisy vs. clean signals (binary) ii) classification with respect to the type of noise (4 classes), iii) classification with respect to SNR level (7 classes) iv) classification with respect to type of noise and SNR level (19 classes).

To achieve this, we employ linear Support Vector Machine (SVM) and SVM with a Radial Basis Function (RBF) kernel, Random Forest (RF) , XGBoost (XGB) , Extremely Randomized Tree (ERT) , Decision Tree (DT) , and Gradient Boosted Decision Trees (GB) [37] classifiers to train our predictive model [32], [35]. We implement these classifiers using scikit-learn with the following parameters.

For the ensemble learning models, including RF, GB, XGB, and Extremely Randomized Trees (ERT), as well as DT, we set the parameters n_estimators, random_state, max_depth, and max_leaf_nodes to 100, 42, 500, and 15, respectively. For linear and RBF SVM, we set random_state, C, and gamma to 42, 0.1, and 0.001, respectively. These parameter settings are chosen to reduce memory usage for storing the models, as compared to using the default settings.

D. Classification Performance Metrics

We employ widely used accuracy and F1-score classification performance metrics for the evaluation of our trained ML models. For multi-class classification, if we have C classes, for accuracy, we simply calculate the proportion of correct predictions to all predictions, and for F1-score, we calculate the weighted average:

$$Weighted\ F1\text{-}score = \frac{\sum_{i=1}^{C} F1\text{-}score_i \times Support_i}{\sum_{i=1}^{C} Support_i}, \quad (1)$$

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}, \quad (2)$$

where $Support_i$ represents the support (the number of true instances) for class i, and $F1\text{-}score_i$ is the F1-score for class i. Additionally, TP, TN, FP, and FN refer to true positives, true negatives, false positives, and false negatives, respectively.

Before training and validating our models, we ensure data balance across all classes. Therefore, the metrics are computed with equal weight for the various classes.

E. Time, Energy, and Memory

For each type of feature, execution time, energy consumption, and memory usage of the steps for inference of a 10 second window of data were measured. We perform all experiments on Linux (Ubuntu 22.04.4 LTS), running on the hardware platform (Table II). We know that the resource metric values are different on a wearable device, which has stricter energy and computational constraints. However, our work provides valuable insights into the relative trade-offs between different ML models and feature types. For example, models or features that are computationally expensive on a laptop are likely to remain so on a wearable device.

We use Python version 3.10.12. We measure time and memory using Python built-in module timeit [38] and tracemalloc [39]. Tracemalloc allows the current memory usage and as well as the peak memory usage to be measured. For energy consumption measurements we use the Running Average Power Limit (RAPL) [40] interface, available on Intel processors, which has been validated by Intel and reported in several papers [41]–[43]. It allows simple monitoring of energy consumption accurately without any power meters. RAPL is split into several power domains such as package,

TABLE II: FEATURES OF HARDWARE PLATFORM.

Feature	Specification
Processor	13th Gen Intel(R) Core (TM) i9-13900 CPU Family 6 Model 183
Clock Frequency	Max: 5.6 GHz Min: 0.8 MHz
Architecture	x86_64
Type	Desktop
Sockets	1
Cores per Socket	24
L3 Cache	36 MiB
Memory	32 GB 32 GB LPDDR4X

energy used by cores, DRAM, and platform. Our platform does not support the DRAM domain. We use the Linux powercap framework [44], which allows the energy counters (in μJ) in each domain of the Intel RAPL to be read with privileged access in Linux.

For different types of classification (Noise, SNR, and Noise-SNR), we measure the time, energy, and current and peak memory used by the cores as well as the platform energy, and report the units in seconds (s or ms), bytes (B), and joules (J or mJ), respectively.

We perform the measurements for each of the following steps: *feature extraction*, *feature normalization*, *loading the classification model*, and *prediction*. In addition, we profile feature extraction for each feature type. We report feature extraction, feature normalization, and prediction resource usages per window and carry all the measurements separately, when no other programs are running.

III. RESULTS AND DISCUSSION

A. Experimental Results

We evaluate the performance of different ML models on the data for each individual feature and for the concatenated features, referred to as concatenated features in various scenarios as follows:

1) *Prediction of the (Type of) Noise*
2) *Prediction of SNR Level*
3) *Prediction of Type of Noise and SNR Level*

The results are shown in Table III. As we can see, using the concatenated features, among the different ML models, XGB outperforms the others with accuracy and F1-score of 67.91% and 67.89% for detection of noise type classes, 44.80% and 44.57% for detection of SNR values classes, and 40.87% and 40.27% for detection of noise-SNR classes. Also, RBF SVM achieves an 80.52% accuracy and an 80.44% F1-score for classifying clean signals versus noisy signals (EM, MA, and BW-corrupted ECG signals) on average. For other feature types, we can still observe that XGB performs better compared to other ML models. This demonstrates the applicability of XGB in the classification of noise-corrupted signals. Statistical features appear to perform better than morphological features, but concatenated features lead to even better accuracy.

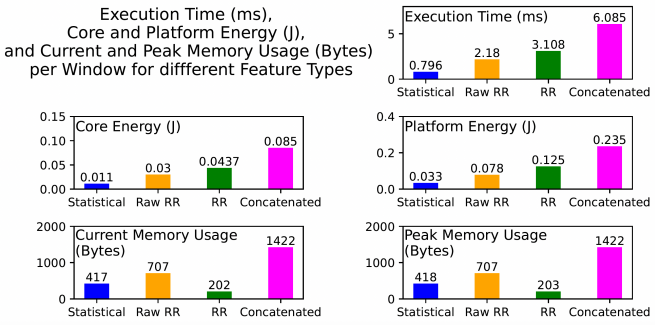

Fig. 1: Time, energy and memory needed for feature extraction per window (same for Noise, SNR, and Noise-SNR classification types) for different type of features. The standard deviations are negligible.

B. Experimental Results for Execution Time, Energy Consumption, and Memory Usage

To perform an inference, we first extract the features, normalize them, and then use the ML model for prediction. The ML model is loaded only once for all inferences of the same classification type. Our goal is to profile resource usage for each step of an inference and analyze it thoroughly.

Feature Extraction per Window: In Figure 1, the gain in accuracy obtained by using concatenated features comes at a cost in comparison to the other features. Table IV presents the time, energy, and memory when using concatenated features, compared to each individual feature type. In this table, as well as in subsequent figures, we report only platform energy since it inherently includes core energy. Similarly, we focus on current memory usage, as it provides more meaningful variations across different ML models and classification classes compared to peak memory usage. However, peak memory usage remains a critical metric for devices with constrained resources. The data in the table highlight the trade-offs of using concatenated features compared to other feature types in terms of time, energy, and memory:

- *Execution Time*: Concatenated features exhibit the highest execution time, taking 7.6 times longer than statistical features and significantly more time compared to RR and Raw RR features.
- *Energy Consumption*: Similarly, concatenated features show the highest energy consumption, consuming 7.2 times more energy than statistical features.
- *Memory Usage*: For memory usage, concatenated features also require the most resources, with 7.0 times higher memory usage compared to RR features.

Despite these costs, concatenated features lead to significant improvements in classification accuracies.

Feature Normalization: For feature normalization, concatenated features result in higher time, energy, and memory requirements compared to other feature types, following a trend similar to that observed during feature extraction. However, there are no significant differences across the classification classes: Noise, SNR, and Noise-SNR. Because the execution time and energy consumption during feature normalization are at least 1000 times smaller than during

TABLE III: PERFORMANCE OF IDNOISE FOR PREDICTION OF DIFFERENT TYPES OF SIGNALS

Feature, Class	Metric	ML Models						
		XGB	**ERT**	**RF**	**Linear SVM**	**RBF SVM**	**GB**	**DT**
Statistical_Noise	**accuracy**	60.16 ± 1.51	54.61 ± 0.92	57.55 ± 1.19	47.53 ± 1.14	43.45 ± 1.00	55.76 ± 0.89	54.36 ± 1.48
	f1_score	60.07 ± 1.46	53.65 ± 0.86	56.99 ± 1.12	46.32 ± 1.13	42.28 ± 0.71	55.18 ± 0.90	54.12 ± 1.35
Statistical_SNR	**accuracy**	38.30 ± 1.70	37.72 ± 1.92	38.38 ± 2.04	29.33 ± 0.88	36.18 ± 0.81	34.96 ± 0.93	31.02 ± 1.17
	f1_score	37.85 ± 1.57	37.60 ± 1.84	38.16 ± 1.96	27.54 ± 0.89	34.45 ± 0.56	33.91 ± 0.72	31.17 ± 1.12
Statistical_Noise-SNR	**accuracy**	32.00 ± 2.07	30.70 ± 2.23	31.60 ± 2.22	23.45 ± 1.52	28.60 ± 1.72	27.96 ± 1.97	24.24 ± 1.33
	f1_score	31.57 ± 2.28	30.34 ± 2.45	31.28 ± 2.47	22.56 ± 1.56	27.92 ± 1.73	27.52 ± 2.10	24.10 ± 1.30
Raw RR_Noise	**accuracy**	53.11 ± 2.20	50.19 ± 1.42	51.45 ± 1.35	41.86 ± 2.23	49.80 ± 2.31	48.93 ± 2.61	41.95 ± 1.13
	f1_score	52.58 ± 2.04	49.50 ± 1.52	51.20 ± 1.46	37.04 ± 1.25	48.56 ± 1.78	47.34 ± 2.02	41.36 ± 1.15
Raw RR_SNR	**accuracy**	33.13 ± 1.04	33.36 ± 0.71	33.54 ± 0.78	24.69 ± 1.17	30.02 ± 1.30	30.94 ± 0.38	25.45 ± 0.87
	f1_score	31.65 ± 0.81	32.99 ± 0.72	33.02 ± 0.80	17.07 ± 0.30	25.31 ± 0.60	27.59 ± 0.03	25.35 ± 0.91
Raw RR_Noise-SNR	**accuracy**	24.55 ± 1.29	26.09 ± 1.47	26.08 ± 1.39	15.21 ± 0.78	19.84 ± 1.14	20.62 ± 1.29	18.62 ± 0.91
	f1_score	22.78 ± 1.61	24.13 ± 1.78	24.29 ± 1.69	10.14 ± 0.83	16.43 ± 1.12	18.95 ± 1.44	18.32 ± 0.91
RR_Noise	**accuracy**	50.93 ± 2.47	45.69 ± 1.69	47.07 ± 1.96	46.80 ± 1.98	51.18 ± 2.00	50.34 ± 2.14	39.58 ± 1.49
	f1_score	50.27 ± 2.20	45.02 ± 1.90	46.88 ± 2.16	44.65 ± 1.12	49.78 ± 1.63	48.71 ± 1.77	38.94 ± 1.60
RR_SNR	**accuracy**	31.85 ± 1.40	29.40 ± 1.38	30.03 ± 1.38	29.17 ± 1.31	31.81 ± 1.27	31.33 ± 1.31	24.13 ± 1.05
	f1_score	29.92 ± 1.03	29.16 ± 1.30	29.65 ± 1.25	22.36 ± 0.82	25.89 ± 0.79	27.50 ± 0.86	24.12 ± 1.02
RR_Noise-SNR	**accuracy**	25.18 ± 1.04	22.51 ± 1.21	23.36 ± 1.22	14.75 ± 0.82	11.61 ± 0.85	21.20 ± 1.21	20.96 ± 1.36
	f1_score	23.72 ± 1.14	19.02 ± 1.25	20.46 ± 1.17	9.21 ± 0.70	5.14 ± 0.52	19.56 ± 1.32	18.78 ± 1.27
Concatenated_Noise	**accuracy**	**67.91 ± 2.35**	63.55 ± 2.46	65.64 ± 2.55	56.37 ± 2.26	65.40 ± 3.08	63.55 ± 2.16	55.00 ± 1.81
	f1_score	**67.89 ± 2.45**	63.19 ± 2.75	65.51 ± 2.71	55.60 ± 2.01	65.32 ± 3.08	63.40 ± 2.10	54.88 ± 1.97
Concatenated_SNR	**accuracy**	**44.80 ± 1.74**	43.80 ± 1.93	44.49 ± 2.24	35.00 ± 1.34	41.23 ± 1.48	40.67 ± 1.54	35.25 ± 0.83
	f1_score	**44.57 ± 1.74**	43.72 ± 2.00	44.31 ± 2.26	33.13 ± 1.26	40.30 ± 1.48	40.02 ± 1.35	35.37 ± 0.88
Concatenated_Noise-SNR	**accuracy**	**40.86 ± 3.39**	33.25 ± 2.77	36.06 ± 3.69	28.29 ± 1.89	14.71 ± 1.06	36.60 ± 2.96	31.85 ± 2.58
	f1_score	**40.27 ± 3.60**	31.97 ± 2.95	35.08 ± 3.95	26.79 ± 1.78	9.38 ± 0.65	36.05 ± 3.17	31.42 ± 2.77
Concatenated_Noise (binary: clean vs. noisy)	**accuracy**	**79.21 ± 3.31**	**66.17 ± 3.43**	**71.63 ± 3.55**	**79.63 ± 3.05**	**80.52 ± 3.11**	**80.46 ± 2.53**	**69.07 ± 2.84**
	f1_score	**79.01 ± 3.54**	**62.13 ± 4.95**	**69.65 ± 4.57**	**79.55 ± 3.00**	**80.44 ± 3.21**	**80.41 ± 2.55**	**67.13 ± 3.64**

* Due to limited space and similar trends to other classification types, we omit results for using other feature types in the binary classification task.

TABLE IV: TIME, ENERGY, AND MEMORY NEEDED FOR FEATURE EXTRACTION PER WINDOW FOR DIFFERENT TYPES OF FEATURES.

Feature Class	Execution Time (ms)	Energy Consumption (mJ)	Memory Usage (B)
Statistical	0.796	33	417
RR	3.108	125	202
Raw RR	2.180	78	707
Concatenated	6.085	235	1422

feature extraction, and memory usage is less than 5 times that of feature extraction. Overall, feature normalization has a negligible impact on resource consumption.

Prediction per Window for each ML Model: The usage metrics for prediction per window for each ML model are displayed in Figure 2. Execution time differences among feature types vary by a factor of up to approximately 2.0 for each ML model. For example, for noise classification in the case of XGB, execution times range from 0.000352 ms (Concatenated) to 0.000523 ms (Raw RR), a factor of 1.49. Similarly, for RBF SVM, the times range from 3.6049 ms (Statistical) to 6.3709 ms (Concatenated), a factor of 1.77. These differences show that execution time across feature types remains reasonably consistent for each model.

For energy consumption per window, the differences among feature types vary by a factor of up to approximately 2.1 for each ML model. For example, for noise classification using RBF SVM, energy consumption ranges from 148.26 mJ (RR) to 290.31 mJ (Concatenated), a factor of 1.96. For lightweight models such as XGB, the variations are smaller, ranging from 41.90 mJ (Statistical) to 83.45 mJ (Raw RR), a factor of 1.72. These observations suggest that differences in energy consumption remain reasonably small between feature types for each ML model.

Memory usage per window remains nearly constant across all feature types for each ML model, with negligible variations, as it depends on the model's structure rather than the input features. In contrast, execution time and energy consumption show slight variations due to differences in feature dimensionality and computational complexity. This effect is most noticeable in SVM models, where predictions involve more floating-point computations due to the kernel function and support vector selection. Although these models load only the relevant support vectors during prediction - reducing memory usage - the increased computational load results in higher time and energy costs.

For each individual inference, the *total time* for feature extraction, feature normalization, and prediction within a 10-second window stays below 20 ms, regardless of feature type, ML model, or classification type. This makes the system well suited for real-time applications. Although the linear SVM and RBF SVM models exhibit the highest execution times and energy consumption, their memory usage is slightly lower than most of the other ML models.

Loading the Model: Although the ML model is typically loaded once for all inferences of the same classification type, we include it in the discussion to account for scenarios where the model is not retained in memory between inferences. Each inference involves feature extraction and normalization, and prediction. The time, energy, and memory required to load an ML model can vary significantly, ranging from over 1000 times to just 10 times that needed for prediction, depending on the feature type and ML model. Consequently, the time, energy, and memory required to load the model cannot always be ignored. For each ML model, the differences in time, energy consumption, and memory usage across different feature types remain within

a factor of 5, regardless of the classification type. In some cases, concatenated features even result in a lower energy consumption. Memory usage shows a consistent pattern across feature types and classification types, with linear SVM and RBF SVM requiring the most memory. Consequently, using concatenated features does not necessarily increase the time, energy, or memory requirements to load the ML model. The key factors in selecting a suitable feature type and ML

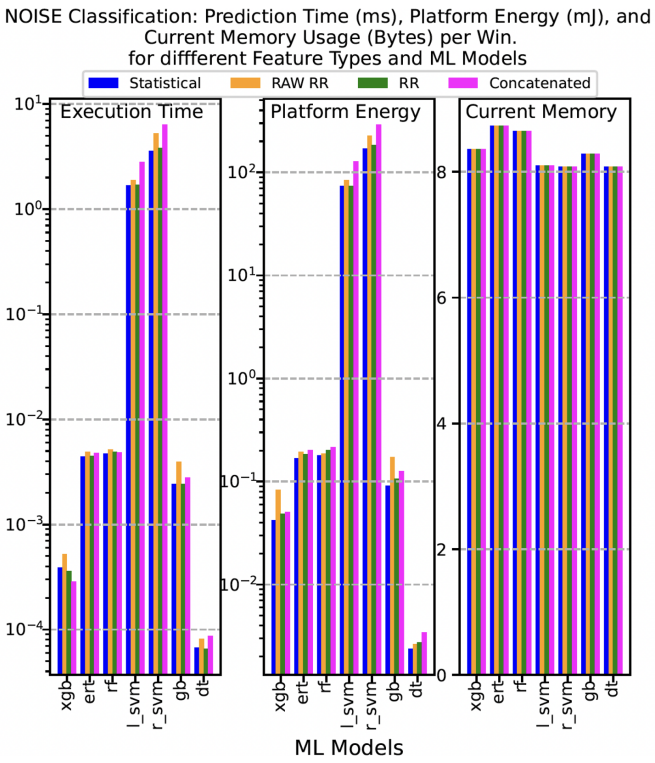

Fig. 2: Usage metrics for prediction per window for different feature types across various ML models for Noise classification. We do not report the plots for SNR and Noise-SNR classifications as they exhibit similar trends.

model include inference accuracy (Table III), feature extraction costs (Table IV) and time, energy, and memory values during both inference and model loading. Balancing accuracy improvements against computational costs is essential for choosing the most suitable solution for a specific application. In addition, energy and memory requirements must align with the constraints of the target hardware platform. To illustrate this decision-making process, we present three scenarios.

Scenario 1: If achieving high accuracy is the top priority, the most suitable ML model for multi-class classification is XGB since it offers the highest accuracy compared to the other ML models, regardless of feature type and classfication type. Specifically, it offers an increase in accuracy for Noise classification from 60.16 for Statistical, 53.11 for Raw RR, 50.93 for RR to 67.91 for Concatenated features. However, choosing concatenated features requires accommodating an increase in execution time, energy consumption, and memory usage for feature extraction. Additionally, if the goal is to perform binary classification for detecting noisy signals regardless of the type of noise, RBF SVM and GB are the most suitable options.

Scenario 2: If accuracy as well as execution time have high priority, we should consider using statistical features (or even other features) and the DT model. This combination offers a 7 to 160 times improvement in execution time compared to other models. Furthermore, DT has the lowest energy consumption and memory usage during both prediction and model loading. Therefore, it is well-suited for applications that require speed, and where limited battery and memory resources can be traded for reduced accuracy.

Scenario 3: If memory usage is the primary concern for a given application, then RR features are preferred, as they have the lowest memory requirements, for feature extraction and prediction. However, the associated accuracy loss varies depending on the feature type, the classification type, and the ML model, with a loss ranging from 17% to 48%.

Concatenated features offer improved accuracy, but this gain must be weighed against the associated costs in execution time, energy consumption, and memory usage. The costs of each step leading to inference per window should be analyzed for each classification class and ML model.

IV. Conclusion and Further Work

Wearable devices are prone to different types of noise. Noise-corrupted signals increase the number of false positives in wearable outputs, leading to decreased user trust. Consequently, true alarms and recommendations from the device may be ignored due to a lack of confidence in its accuracy, which can threaten the user's health. In this paper, we proposed a noise type and SNR level detection method for ECG signals, allowing the identification of clean ECG signals versus 3 different noise types that corrupt the signal at 6 SNR levels for each noise type. To achieve this, we used statistical and morphological features, as well as their concatenated forms, to train various ML models. By distinguishing between clean and noisy signals, we can reduce false positive alarms in wearable devices, thereby improving the functionality of the devices.

We find that, although concatenated features deliver better accuracy, they lead to increased execution time, energy consumption, and memory usage during feature extraction phase. During the prediction, concatenated features do not lead to significant changes in time and resource consumption for the Noise, SNR, and Noise-SNR classification types and the considered set of ML models. With the help of the scenarios, we illustrate the possible trade-offs involved in selecting the feature type and ML model for a given application and/or target hardware with given execution time, energy, and memory constraints.

These results offer a solid baseline for selecting efficient ML models and feature sets when transitioning to wearable platforms. The next step in this work is to investigate more possibilities to improve accuracy, execution time, energy consumption, and memory usage. This will involve adapting and implementing current algorithms on resource-constrained hardware platforms.

REFERENCES

[1] P. Lukowicz *et al.* Wearable systems for health care applications. *Methods of information in medicine*, 43(03):232–238, 2004.

[2] P. Traunmuller *et al.* Wearable healthcare devices for monitoring stress and attention level in workplace environments, 2024.

[3] D. Sopic *et al.* Personalized seizure signature: An interpretable approach to false alarm reduction for long-term epileptic seizure detection. *Epilepsia*, 2022.

[4] X. Erickson *et al.* Personalized seizure detection using spiking neural networks. In *2023 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, pp. 1–6. IEEE, 2023.

[5] D. Sopic *et al.* Real-time classification technique for early detection and prevention of myocardial infarction on wearable devices. In *2017 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pp. 1–4. IEEE.

[6] M. Götzinger *et al.* Enhancing the early warning score system using data confidence. In P. Perego *et al.*, editors, *Wireless Mobile Communication and Healthcare*, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pp. 91–99. Springer International Publishing.

[7] M. Götzinger *et al.* Confidence-enhanced early warning score based on fuzzy logic. 27(2):691–708.

[8] N. TaheriNejad. Wearable medical devices: Challenges and self-aware solutions. In *IEEE Life Sciences*, volume 2, pp. 5–6, April 2019.

[9] N. TaheriNejad *et al.* Mobile health technology: From daily care and pandemics to their energy consumption and environmental impact. *ACM/Springer Mobile Networks and Applications*, pp. 1–5, 2022.

[10] L. Xie *et al.* Computational diagnostic techniques for electrocardiogram signal analysis. *Sensors*, 20(21):6318, 2020.

[11] K. Guk *et al.* Evolution of wearable devices with real-time disease monitoring for personalized healthcare. *Nanomaterials*, 9(6):813, 2019.

[12] O. Ayanwale. *Impact Evaluation of Wearable Devices on Physical Activity and Healthy Lifestyles*. PhD thesis, Hochschule für Angewandte Wissenschaften Hamburg, 2022.

[13] SMARTsemi. Optimizing memory architectures for wearable health devices, Aug 2024. Accessed on 07.08.2024.

[14] C. C. Dobrescu *et al.* Direct memory access-based data storage for long-term acquisition using wearables in an energy-efficient manner. *Sensors*, 24(15):4982, 2024.

[15] S. Mohamed *et al.* Energy management for wearable medical devices based on gaining–sharing knowledge algorithm. *Complex & Intelligent Systems*, 9(6):6797–6811, 2023.

[16] E. Hadizadeh *et al.* A low-power signal-dependent sampling technique: Analysis, implementation, and application. *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–14, 2020.

[17] G. B. Moody *et al.* A noise stress test for arrhythmia detectors. *Computers in cardiology*, 11(3):381–384, 1984.

[18] S. Mian Qaisar. Baseline wander and power-line interference elimination of ecg signals using efficient signal-piloted filtering. *Healthcare technology letters*, 7(4):114–118, 2020.

[19] T. Alkhidir *et al.* Simple method for adaptive filtering of motion artifacts in e-textile wearable ecg sensors. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 3807–3810, 2015.

[20] Z. Zhang *et al.* Adaptive motion artefact reduction in respiration and ecg signals for wearable healthcare monitoring systems. *Medical & biological engineering & computing*, 52:1019–1030, 2014.

[21] D. Tong *et al.* Adaptive reduction of motion artifact in the electrocardiogram. In *Proceedings of the Second Joint 24th Annual Conference and the Annual Fall Meeting of the Biomedical Engineering Society][Engineering in Medicine and Biology*, volume 2, pp. 1403–1404. IEEE, 2002.

[22] X. An *et al.* Adaptive motion artifact reduction in wearable ecg measurements using impedance pneumography signal. *Sensors*, 22(15):5493, 2022.

[23] S. Nagai *et al.* Motion artefact removals for wearable ecg using stationary wavelet transform. *Healthcare technology letters*, 4(4):138–141, 2017.

[24] S. Abbaspour and A. Fallah. Removing ecg artifact from the surface emg signal using adaptive subtraction technique. *Journal of biomedical physics & engineering*, 4(1):33, 2014.

[25] N. E. Huang *et al.* The empirical mode decomposition and the hilbert spectrum for nonlinear and non-stationary time series analysis. *Proceedings of the Royal Society of London. Series A: mathematical, physical and engineering sciences*, 454(1971):903–995, 1998.

[26] C. Zou *et al.* Motion artifact removal based on periodical property for ecg monitoring with wearable systems. *Pervasive and Mobile Computing*, 40:267–278, 2017.

[27] U. Satija *et al.* A simple method for detection and classification of ecg noises for wearable ecg monitoring devices. In *2015 2nd International Conference on Signal Processing and Integrated Networks (SPIN)*, pp. 164–169. IEEE, 2015.

[28] A. L. Goldberger *et al.* Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. *circulation*, 101(23):e215–e220, 2000.

[29] G. B. Moody and R. G. Mark. The impact of the mit-bih arrhythmia database. *IEEE engineering in medicine and biology magazine*, 20(3):45–50, 2001.

[30] G. B. Moody *et al.* The MIT-BIH noise stress test database.

[31] Waveform database.

[32] S. Khooyooz *et al.* A novel machine-learning-based noise detection method for photoplethysmography signals. In *46th Annual International Conference of the IEEE Engineering in Medicine Biology Society (EMBC)*. IEEE, 2024.

[33] R. Krishnan *et al.* Analysis and detection of motion artifact in photoplethysmographic data using higher order statistics. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 613–616. IEEE, 2008.

[34] V.-K. D. Le *et al.* vital_sqi: A python package for physiological signal quality control. *Frontiers in Physiology*, 13:1020458, 2022.

[35] A. Aminifar *et al.* Recognoise: Machine-learning-based recognition of noisy segments in electrocardiogram signals. In *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2024.

[36] P. M. Caridade Gomes. *Development of an open-source Python toolbox for heart rate variability (HRV)*. PhD thesis, Hochschule für angewandte Wissenschaften Hamburg, 2019.

[37] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232, 2001.

[38] Python Software Foundation. Python 3.10.12 documentation, timeit. https://docs.python.org/3/, 2024. [visited on 19.09.2024].

[39] Python Software Foundation. Python 3.10.12 documentation, tracemalloc. https://docs.python.org/3/, 2024. [visited on 19.09.2024].

[40] Intel Corporation. *Intel 64 and IA-32 Software Developer Manuals - Volume 3*, 2024. Available at https://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html, [visited on 19.09.2024].

[41] K. N. Khan *et al.* Rapl in action: Experiences in using rapl for power measurements. 3(2), mar 2018.

[42] H. David *et al.* Rapl: Memory power estimation and capping. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 189–194, 2010.

[43] M. Hähnel *et al.* Measuring energy consumption for short code paths using rapl. In *Proceedings of the Conference/Journal Name*, pp. pages, Dresden, Germany, Year. Publisher.

[44] The Linux Kernel Documentation Project. Power Capping Framework. https://www.kernel.org/doc/html/latest/power/powercap/powercap.html, 2024. [visited on 19.09.2024].